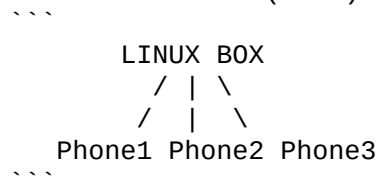


Mesh Topology Vision: Phone-to-Phone Bridging

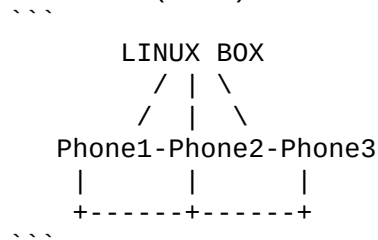
The Goal

Extend the star-bridge model to allow **peer discovery and direct SSH tunneling** between phones

Current Model (Star):



Vision (Mesh):



Each phone can reach others directly via SSH, forming a resilient network with no central cloud

Core Principles

1. **Zero Commercial Platforms** – Pure SSH, routing tables, and custom code
2. **Pure P2P Discovery** – Phones find each other via local scan or config
3. **SSH Tunneling** – All bridges built on SSH's native capabilities
4. **Resilient Chains** – If direct link fails, reroute through intermediaries
5. **Simple Implementation** – Code must be understandable; favor clarity over optimization
6. **Minimal Exceptions** – Only peer discovery service allowed; everything else native code

Architecture

Phase 1: Phone Discovery (Minimal Exception)

Phones need to find each other. Options:

Option A: Local Network Scanning (Pure)

```

python
def discover_phones_on_network(prefix="192.168.44"):
    """Scan local subnet for responding SSH daemons."""
    discovered = []
    for i in range(1, 255):
        ip = f"{prefix}.{i}"
        if ssh_probe(ip, port=8022, timeout=1): # Non-blocking probe
            discovered.append(ip)
    return discovered

```

Pros: No external service needed

Cons: Slow, noisy, works only on same subnet

****Option B: Lightweight Registry (Acceptable Exception)****

A simple file-based registry on Linux that stores known phones:

```
...\n~/.config/star-bridge/known_phones.json\n{\n  "phone1": {"ip": "192.168.44.1", "hostname": "pixel6", "last_seen": "2026-02-06T15:30:00Z"},\n  "phone2": {"ip": "192.168.44.2", "hostname": "samsung", "last_seen": "2026-02-06T15:31:00Z"}\n}...\n
```

Each phone, when it connects via `ssh_admin_connection.py`, updates this registry.

Recommended: ****Start with Option B**** (1 service, pure code for everything else)

Phase 2: Phone-to-Phone SSH Tunneling

Once phones know each other, establish direct SSH bridges:

```
```python\ndef bridge_phones(phone_a_ip, phone_b_ip, username, service_port=5001):\n    """\n    Create SSH tunnel: Phone A → Phone B (for service on Port)\n\n    Allows: Linux → Phone A → Phone B services\n    """\n    # On Phone A, create reverse tunnel to Phone B\n    cmd = f"""\nssh -N -R {service_port}:localhost:{service_port} \\\n    {username}@{phone_b_ip} -p 8022 \\\n    -o StrictHostKeyChecking=no\n    """\n    # Now: Phone A can reach Phone B's service via localhost:5001\n    return subprocess.Popen(cmd, shell=True)\n...\n
```

**\*\*Result:\*\*** Phone A → (SSH bridge) → Phone B:5001

### **### Phase 3: Routing & Failover**

Phones maintain a **\*\*routing table\*\*** mapping services to reachable nodes:

```
```python\nclass PhoneMesh:\n    def __init__(self):\n        self.routes = {\n            "phone1:5001": {\n                "direct": "192.168.44.1:5001",\n                "via_phone2": "192.168.44.2 → 192.168.44.1:5001",\n                "via_phone3": "192.168.44.3 → 192.168.44.2 → 192.168.44.1:5001",\n            }\n        }\n\n    def get_best_path(self, target_service):\n        """Return shortest path to target service."""\n        paths = self.routes[target_service]\n
```

```

    # Prefer direct, fallback to via_phone2, then via_phone3
    for path in ["direct", "via_phone2", "via_phone3"]:
        if self.is_reachable(paths[path]):
            return paths[path]
    return None

def is_reachable(self, path):
    """Probe the path without consuming bandwidth."""
    # Quick SSH connectivity check
    return ssh_probe(path, timeout=2)
...

---

## 90s P2P Inspiration

This approach mirrors Gnutella and Kazaa from the late 90s:

1. Local node discovery – Nodes announce themselves on LAN
2. Peer directories – Simple rosters of known peers
3. Chain routing – Find content by asking neighbors
4. No central authority – Fully decentralized

Applied here:

- Phones are nodes
- Services (web_server, auth_server) are resources
- SSH tunnels are query paths
- Registry is the peer directory

---

## Pseudo-Code: Full Mesh Connection

...

SYSTEM STATE:
linux_box$ Has ssh_admin_connection.py running
phone1$ Connected to linux via USB/hotspot
phone2$ Connected to linux via USB/hotspot

STEP 1: Phones report to registry
phone1$ python ssh_admin_connection.py
■■→ Updates ~/.config/star-bridge/known_phones.json
{ "phone1": { "ip": "192.168.44.1", ... } }

phone2$ python ssh_admin_connection.py
■■→ Updates registry
{ "phone2": { "ip": "192.168.44.2", ... } }

STEP 2: Linux reads registry and builds mesh
linux$ read known_phones.json
linux$ for each phone_pair:
    establish_bridge(phone_a, phone_b)

STEP 3: Phone-to-Phone Bridge
linux$ ssh -N -R 5001:localhost:5001 \
    u0_aXXX@192.168.44.2 -p 8022

Result: Phone1's service now accessible from Phone2

```

STEP 4: Failover

If Phone1 → direct fails:

Try: Linux → Phone2 → Phone1 (chain tunnel)

Try: Linux → Phone3 → Phone1 (if available)

STEP 5: Service Discovery

Phone1\$ curl http://localhost:5001/

↓ (local service)

Phone1\$ curl http://phone2:5001/

↓ (via SSH tunnel from Step 3)

...

Exception Cases

1. **Peer Discovery** – Registry service or local scan (unavoidable)
2. **Distance routing** – May need NAT/firewall workarounds (log as known issues)
3. **Bandwidth constraints** – Document limitations

Everything else: pure SSH + custom Python code.

Implementation Phases

Phase 1 (Now): Single phone + Linux (current star-bridge)

Phase 2 (Next): Two phones + registry

- Add known_phones.json registry
- Implement phone_discovery() in ssh_admin_connection.py
- Document manual registry updates

Phase 3 (Future): Auto-bridging

- Extend ssh_admin_connection.py to establish phone-phone tunnels
- Implement routing table logic
- Add failover detection

Phase 4 (Future): Service discovery API

- Phones advertise services to registry
- Other phones query and auto-connect

Code Organization

...

star-bridge/

■■■ ssh_admin_connection.py	(current: Linux ↔ Phone)
■■■ mesh_registry.py	(Phase 2: Registry manager)
■■■ mesh_router.py	(Phase 3: Routing + failover)
■■■ phone_discovery.py	(Phase 2: Find peers)
■■■ auth-server/	
■■■ auth_server.py	(unchanged)

...

Key Insight

The **mesh** doesn't require a new protocol. It's just SSH doing what it was designed to do:

- **SSH tunnel** = bridge between two endpoints
- **Chained tunnels** = bridge through intermediary
- **Local routing table** = which path to use

Pure code, pure concepts, 90s-proven reliability.